Parallel *In Situ* Detection of Connected Components in Adaptive Mesh Refinement Data

Xiaocheng Zou[1,2], Kesheng Wu[3,*], David A. Boyuka II[1,2], Daniel F. Martin[3], Suren Byna[3], Houjun Tang[1,2], Kushal Bansal[1,2], Terry J. Ligocki[3], Hans Johansen[3], and Nagiza F. Samatova[1,2,*]
[1] North Carolina State University, NC 27695, USA
[2] Oak Ridge National Laboratory, TN 37831, USA
[3] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[*] Corresponding authors: kwu@lbl.gov, samatova@csc.ncsu.edu

***Abstract*—Adaptive Mesh Refinement (AMR) represents a significant advance for scientific simulation codes, greatly reducing memory and compute requirements by dynamically varying simulation resolution over space and time. As simulation codes transition to AMR, existing analysis algorithms must also make this transition. One such algorithm, *connected component detection*, is of vital importance in many simulation and analysis contexts, with some simulation codes even relying on *parallel, in situ* connected component detection for correctness. Yet, current detection algorithms designed for uniform meshes are not applicable to hierarchical, non-uniform AMR, and to the best of our knowledge, AMR connected component detection has not been explored in the literature. Therefore, in this paper, we formally define the general problem of connected component detection for AMR, and present a general solution.**

Beyond solving the general detection problem, achieving viable *in situ* detection performance is even more challenging. The core issue is the conflict between the communication-intensive nature of connected component detection (in general, and especially for AMR data) and the requirement that *in situ* processes incur minimal performance impact on the co-located simulation. We address this challenge by presenting the first connected component detection methodology for structured AMR that is applicable in a parallel, *in situ* context. Our key strategy is the incorporation of an multi-phase AMR-aware communication pattern that synchronizes connectivity information across the AMR hierarchy.

In addition, we distill our methodology to a generic framework within the Chombo AMR infrastructure, making connected component detection services available for many existing applications. We demonstrate our method's efficacy by showing its ability to detect ice calving events in real time within the real-world BISICLES ice sheet modeling code. Results show up to a 6.8x speedup of our algorithm over the existing specialized BISICLES algorithm. We also show scalability results for our method up to 4,096 cores using a parallel Chombo-based benchmark.

I. Introduction

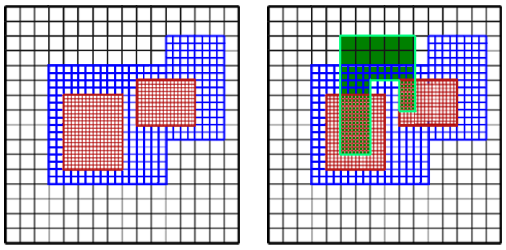
One of the most significant advances for large-scale scientific simulations has been the advent of Adaptive Mesh Refinement, or AMR [1]–[3]. By dynamically refining simulation resolution across space and time, AMR simulation codes can drastically improve efficiency of computational resources while meeting or exceeding acceptable error levels for numerical accuracy. The result of this refinement is a hierarchical, multi-level, and multi-resolution mesh (Fig. 1(a)), which gives rise to many opportunities and challenges in data analytics and visualization due to the intricate mesh structure.

As simulations transition to AMR, existing analysis and support algorithms must become AMR-aware to match. Connected component detection is one such algorithm, and is important to many scientific applications in both *in situ* (at simulation run time) and *post-processing* contexts. For example, the BISICLES AMR ice sheet modeling code [4] relies on the correctness-critical task of real-time isolated iceberg detection, which can be solved with efficient *in situ* connected component detection. Likewise, in the context of post-analysis, operations such as isosurfacing, identification of regions of interest, and hot spot isolation are often transformed into connected component detection tasks.

Developing a connected component detection algorithm for AMR data (Fig. 1(b)) is not a trivial task, however. Existing detection algorithms, generally categorized as one-pass [5]–[7], two-pass [8]–[11], and multi-pass [12], are designed for single-level, uniform meshes, and thus are not applicable to data that span multiple levels in an adaptive refinement mesh. Attempting to "flatten" an AMR hierarchy to a uniform mesh (refining all mesh levels up to the finest resolution) in order to apply existing algorithms has its own problems: such an operation results in an explosive increase in memory usage, which is untenable for extreme-scale datasets, and in any case defeats one primary goal of using AMR in the first place.

To the best of our knowledge, AMR connected component detection has not been well-defined in the literature. Therefore, in this paper, we formally define the general problem of connected component detection that is specifically tailored for AMR data (Section II). By adapting the traditional definitions of mesh, adjacency, and connectivity, we clearly articulate the problem to be solved, which we expect will benefit future work in this area. We then describe our solution to the AMR connected component detection problem in Section III-A.

Beyond the general problem of AMR-aware connected component detection, achieving viable *in situ* detection is a challenge. Any *in situ* processing must have minimal performance impact on the overall simulation. Yet, parallel connected component detection is inherently characterized as communication-intensive, and the non-uniform, distributed nature of AMR data exacerbates this state of affairs. Specifically, the distribution of mesh data across processes in AMR sim-



(a) (b)

Fig. 1. *Subfigure (a)* shows an AMR mesh with three levels: the black mesh is the coarsest mesh (level 0), the blue mesh is a finer mesh (level 1), and the red mesh is the finest (level 2). The refinement ratio equals 2. *Subfigure (b)* shows a connected component that has been detected, spanning all three AMR levels.

ulations is typically both *non-regular* and *dynamic over time*, making synchronization of connectivity information inherently more difficult than with the regular domain decompositions applied to uniform meshes.

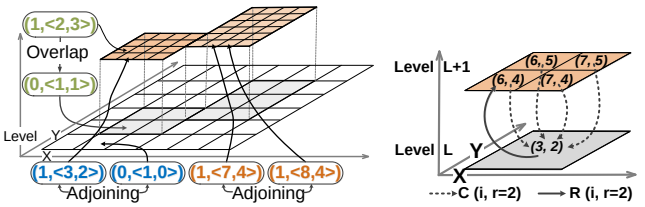
To address this challenge, we present the first connected component detection methodology for structured AMR data that is applicable in a distributed, *in situ* context (Section III-B). Our key strategy is to use a parallel, *in situ*, AMR-aware communication method to synchronize component connectivity across the AMR structure distributed over many processes. By using a hierarchical process grouping conforming to the AMR refinement level structure, we are able to compute global connectivity over arbitrary distributions of AMR data without expensive all-to-all communication.

In addition, we distill our parallel *in situ* approach to a general-purpose AMR-aware connected component detection framework in the Chombo infrastructure [13], a popular block-structured AMR framework used by many scientific applications [4], [14], [15]. In this way, we demonstrate the general applicability of our work to scientific codes (Section IV).

Finally, we demonstrate the efficacy of our connected component detection framework when applied to two simulation use cases. We use our framework to provide real-time ice calving detection for BISICLES, a large-scale AMR code, demonstrating viability for *in situ* use (Section IV-B). Additionally, we apply our algorithm to the established "Packed Channel" benchmark [16], [17], for which our algorithm exhibits scalability up to 4,096 cores (Section IV-C).

II. Problem Statement

The ideal that most scientific simulations approximate is the evolution of *dependent variables* (or *fields*), representing physical quantities over a *continuous* spatial domain and over time. Under this ideal, the *connected component detection* problem is to find, at a given instant in time, the maximal connected regions in this continuous domain where these dependent variables meet some criteria. This process yields "interesting regions" representing important information when the criteria is based on domain-specific knowledge. For example, in the BISICLES code [4], connected component detection can identify isolated sections of ice by using a criterion such as



(a) (b)

Fig. 2. Subfigure (a) shows a two-dimensional example of an AMR hierarchy with refinement ratio 2 and two levels. It also indicates pairs of adjoining cells on the same level and on different levels, as well as a pair of overlapping cells. Subfigure (b) illustrates the coarsening and refinement operators at refinement ratio 2, as defined in Definition 1 of the Problem Statement.

$(\text{ice thickness}) > \textit{threshold}$, yielding key information needed to progress the simulation.

Of course, using a continuous domain is not possible with a finite computer, and so simulations necessarily operate on discretized domains. Although uniform meshes have traditionally been used for this purpose, Adaptive Mesh Refinement (AMR) has proven to be a memory- and computation- efficient alternative. AMR assigns different levels of resolution across the simulation domain, delivering enough precision in highly-dynamic areas to achieve the desired accuracy, while also not over-provisioning more static regions.

This paper presents the first general-purpose method for connected component detection *for AMR meshes*, operating under the additional constraints of a *parallel, in situ* context. To clarify, we present a formal definition of connected component detection on AMR data. Then, we restate the problem using this formal language.

First, we define the *coarsening* and *refinement* operators, which will be helpful later[1]:

***Definition** 1:* The *coarsening operator* $C : \mathbb{Z}^D \to \mathbb{Z}^D$ is defined as $C(\boldsymbol{i}, r) = (\lfloor \frac{i_0}{r} \rfloor, \lfloor \frac{i_1}{r} \rfloor, \ldots, \lfloor \frac{i_{D-1}}{r} \rfloor)$, where $r \in \mathbb{Z}^+$ is a given *refinement ratio*. Correspondingly, we define the opposite *refinement operator* $R : \mathbb{Z}^D \to \mathscr{P}(\mathbb{Z}^D)$ as $R(\boldsymbol{i}, r) = \{\boldsymbol{i}' \in \mathbb{Z}^D \mid C(\boldsymbol{i}', r) = \boldsymbol{i}\}$. Equivalently, $R(\boldsymbol{i}, r) = \prod_{j=0}^{D-1} \{ri_j, ri_j + 1, \ldots, ri_j + r - 1\}$, where $\prod$ is the N-ary Cartesian product. These operators can be naturally extended to operate on sets of vectors.

Next, we define the components of an AMR structure.

***Definition** 2:* An *AMR structure* Ω with $N \in \mathbb{Z}^+$ levels, domain bound $\boldsymbol{u} \in (\mathbb{Z}^+)^D$, and refinement ratio $r \in \mathbb{Z}^+$ consists of a list of *AMR levels* $\Omega_0, \ldots, \Omega_{N-1}$. $\Omega_0 = \{\boldsymbol{i} \in \mathbb{Z}^D \mid \boldsymbol{0} \leq \boldsymbol{i} < \boldsymbol{u}\}$ refers to the first level, with $\boldsymbol{0}$ denoting the zero vector and with $<$ ($\leq$) denoting the pairwise comparison of the components of two vectors. For subsequent levels, $\Omega_{l+1} \subset R(\Omega_l, r)$. Finally, every Ω_l can be decomposed into a disjoint union of m_l rectangular *boxes* $B_{l,k} \subset \mathbb{Z}^D$ such that $\Omega_l = \cup_{k=0}^{m_l} B_{l,k}$ and $B_{l,x} \cap B_{l,y} = \emptyset$ when $x \neq y$.

In other words, an AMR structure is a hierarchy of meshes, with the coarsest (lowest) level covering the whole computational domain, and successively finer (higher) levels covering

[1]Definitions 1 and 2 are adapted from the Chombo framework design document [13]. For brevity, we have simplified these definitions to only those elements pertinent to our specific problem.

portions of the next-coarser level. All mesh cells on a given level have the same spacing, with the grid spacing at each finer level reduced by a given *refinement ratio* relative to the next-coarser level. Additionally, in the specific case of *block-structured AMR* considered in this paper, each level's mesh can always be decomposed in a set of non-overlapping rectangular regions called *boxes*. Figure 2 sums up this definition with an example of a block-structured AMR hierarchy.

***Definition** 3:* A *cell* at level l of AMR structure Ω is defined as a pair $c = (l, \boldsymbol{i})$ with $\boldsymbol{i} \in \Omega_l$. The set of all cells in an AMR structure Ω is denoted by $\sigma(\Omega) = \bigcup_{l=0}^{N-1} (l, \boldsymbol{i}) \mid \boldsymbol{i} \in \Omega_l$.

Now, we define some properties of AMR cells.

***Definition** 4:* Given two cells $c_1 = (l_1, \boldsymbol{i}_1), c_2 = (l_2, \boldsymbol{i}_2)$ in AMR structure Ω, we define the *adjoining predicate* $A : \sigma(\Omega) \times \sigma(\Omega) \to \{true, false\}$ as

$$A(c_1, c_2) = \begin{cases} \boldsymbol{i}_1 - \boldsymbol{i}_2 \in E & \text{if } l_1 = l_2 \\ \exists\, \boldsymbol{j} \in R\left(\boldsymbol{i}_2, r^{l_1 - l_2}\right)\ A((l_1, \boldsymbol{i}_1), (l_1, \boldsymbol{j})) & \text{if } l_1 > l_2 \\ A(c_2, c_1) & \text{else} \end{cases}$$

with $E = \{e_0, -e_0, \ldots, e_{D-1}, -e_{D-1}\}$ the set of unit vectors aligned with a coordinate axis (such as $(1,0,0), (-1,0,0), (0,1,0)$). Cells c_1 and c_2 are said to be *adjoining* iff $A(c_1, c_2)$.

Informally, two adjoining cells would be touching if the AMR hierarchy were "collapsed" or "flattened".

***Definition** 5:* Given two cells $c_1 = (l_1, \boldsymbol{i}_1), c_2 = (l_2, \boldsymbol{i}_2)$ in AMR structure Ω, we define the *overlap predicate* $B : \sigma(\Omega) \times \sigma(\Omega) \to \{true, false\}$ as

$$B(c_1, c_2) = \begin{cases} \boldsymbol{i}_1 = \boldsymbol{i}_2 & \text{if } l_1 = l_2 \\ \boldsymbol{i}_1 \in R\left(\boldsymbol{i}_2, r^{l_1 - l_2}\right) & \text{if } l_1 > l_2 \\ B(c_2, c_1) & \text{else} \end{cases}$$

Cells c_1 and c_2 are said to *overlap* iff $B(c_1, c_2)$.

***Definition** 6:* Given an AMR structure Ω, a *cell-classifying predicate* $\psi : \sigma(\Omega) \to \{true, false\}$ classifies each cell as either a *foreground* ($true$) or *background* ($false$) cell. When ψ is understood by context, the set of foreground and background cells of Ω under ψ are denoted as Ω_{FG} and Ω_{BG}, respectively.

Note, ψ is typically is based on one or more *dependent variables* defined over the cells in Ω. Going back to our earlier example, BISICLES might choose ψ as a threshold on the ice thickness variable, thus identifying icebergs (foreground cells) disconnected from the main ice sheet by regions of thin or no ice (background cells).

Finally, we bring together previous concepts to define "connectedness" and "connected components".

***Definition** 7:* Given an AMR structure Ω with cell-classifying predicate ψ defining foreground cells Ω_{FG}, we define the *adjacency relation* Δ on the foreground cells as $\Delta = \{(c_1, c_2) \in \Omega_{FG} \times \Omega_{FG} \mid A(c_1, c_2) \wedge \neg B(c_1, c_2)\}$.

***Definition** 8:* Given the adjacency relation Δ, we define the *connectivity relation* Δ^+ as the *transitive closure* of Δ. Because Δ^+ is an equivalence relation, it partitions Ω_{FG} into a set of connected components $P = \{\rho_1, \ldots, \rho_{|P|}\}$ with $\rho_x \cap \rho_y = \emptyset \leftrightarrow x \neq y$ and $\Omega_{FG} = \cup_{\rho \in P} \rho$. We say two cells $c_1, c_2 \in \Omega_{FG}$ are *connected* iff $(c_1, c_2) \in \Delta^+$.

That is, two foreground cells are connected if there exist zero or more other foreground cells that form a path of adjacent pairs (under adjacency relation Δ) between them. The resultant set of connected components P is a set of "regions" of cells, within which any two cells are connected.

We can now rephrase the problem statement more formally:

***Definition** 9:* Given an adaptive refinement mesh Ω and a cell-classifying predicate ψ, detect the set of connected components P of foreground cells Ω_{FG} under the connectivity relation Δ^+.

Beyond this basic definition, in order to feasibly run *in situ* with real-world simulations, any connected component detection method must meet two additional constraints. First, in order to operate *in parallel*, Ω must be assumed to be distributed across multiple parallel processes. Second, when running *in situ* (that is, concurrent with a simulation), the solution must have limited runtime disturbance to the simulation (i.e., computation and communication overhead).

III. Method

Returning to the contributions for this paper, we propose a methodology that achieves two key goals, covered individually in the following subsections. First, we show how to solve the connected components detection problem for AMR data (as defined in the previous section); our key insights that make this possible are discussed in Section III-A. Second, we show how to extend these principles to a parallel, in situ context, focusing on minimizing communication and maximizing parallelism by using an AMR-aware communication pattern, discussed in Section III-B.

A. Connected Component Labeling for AMR Data

The central challenge in detecting connected components for AMR data is that connected components may span multiple AMR levels and boxes. Previous detection algorithms for uniform grids rely on scanning the entire dataset one or more times, examining the neighbors of each visited cell to build up connectivity information. However, with hierarchical, non-uniform AMR, these operations are not straightforward. For instance, there is no obvious traversal order over a non-uniform AMR structure, nor is it clear that existing neighbor-masking methods would produce correct results for a given traversal order.

Our key insight is to depart from scanning the AMR hierarchy as a single, indivisible mesh. Instead, we embrace the AMR structure by detecting components within each AMR box separately, followed by joining these components in a global context. This has the advantage of a regular access pattern admitted by the individual uniform-mesh boxes.

This strategy is captured in Algorithm 1, which can be broken down into two phases. *Phase I* (lines 2 to 7) detects connected components within each level. Following this, *Phase II* (lines 9 to 12) then joins these intra-level components across levels, forming finalized global components.

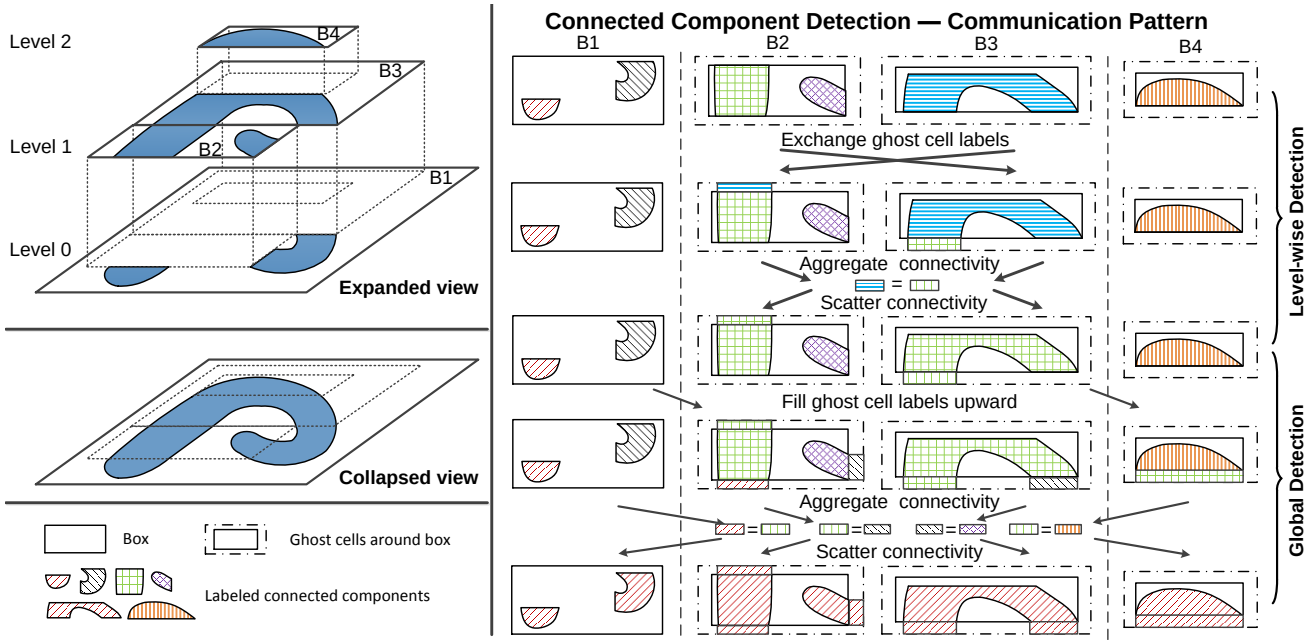


Fig. 3. An example showing the communication pattern in our parallel, *in situ* connected component detection over four AMR boxes. On the left, the component being detected is illustrated with both expanded and collapsed views.

As a preliminary detail, we briefly review the SAUF [8] algorithm, as we utilize this algorithm in some parts of our AMR detection solution. SAUF is a two-pass labeling algorithm with an array-based *union-find* structure. It operates on a uniform, rectangular grid, and employs a cell-classifying predicate ψ to divide cells into foreground and background groups (the same as in our problem statement). The *first pass* in SAUF assigns provisional labels to each cell in raster order (Fig. 4). During the pass, label equivalences are recorded in an array-based *union-find* structure (denoted UF). In the *second pass*, every provisional label is replaced by its corresponding final label as computed in the UF.

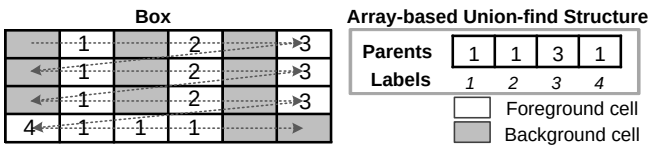


Fig. 4. An example showing provisional labels produced by the first pass of the SAUF algorithm on a 4×6 box (left), along with the corresponding union-find structure UF (right). The dashed arrow indicates the scan traversal order (raster order). The UF structure indicates that labels 1, 2, and 4 are equivalent, and thus collectively form a component (label 3 represents a second component).

Phase I: In this first phase, each level of the AMR hierarchy is considered separately. For a given level ℓ, we first identify *provisional* components within the level by performing the first pass of the SAUF algorithm on each AMR box. This invocation yields a union-find structure UF_ℓ and provisional component labels L_ℓ for each box (line 3). To maintain a global label space, a next label counter s is kept. Each levels labeling starts at s (line 3), and updates s afterward (line 6). All

Algorithm 1: General connected component detection algorithm for AMR data

Input : N-level AMR data, Ω
Input : Cell-classifying predicate, ψ
Output: Labelled AMR data, L

1 `//Phase I`
2 **for** $\ell = \{0, \ldots, N-1\}$ **do**
3 $\quad UF_\ell,\ L_\ell = labeling\_boxes(s, \Omega_\ell, \psi)$
4 $\quad LP_\ell = collect\_label\_equiv\_on\_box\_boundary(L_\ell)$
5 $\quad UF'_\ell = union\_label\_equivs(LP_\ell, UF_\ell)$
6 $\quad s = s + |UF'_\ell|$
7 $\quad GUF = GUF \bigoplus UF'_\ell$
8 `//Phase II`
9 **for** $\ell = \{1, \ldots, N-1\}$ **do**
10 $\quad LP'_\ell = collect\_label\_equiv\_across\_levels(L_\ell, L_{\ell-1})$
11 $\quad GUF' = union\_label\_equivs(GUF, LP'_\ell)$
12 $update\_all\_labels(L_\ell, GUF')$

level-wise union-find structures are concatenated as they are generated, building a global union-find structure (line 7). Having identified box-local components, the algorithm proceeds to connect components across the whole level. It first walks the boundary of each box, building a list of label equivalence pairs between the boundary cells of the current box and those of any neighboring boxes (line 4). The labels from the resultant equivalence pairs are then merged in the union-find structure UF_ℓ, connecting components between boxes (line 5).

Phase II: At this point, box-local components have been captured and joined into level-wise components, as described by the associated global union-find structure. However, larger connected components spanning multiple AMR levels may

still exist. For example, in Figure 3, a component in box "B2" on AMR level 1 (marked in purple color) and a component in box "B3" on AMR level 1 (marked in green color) belong to the single global connected component as they are bridged via the black (rightmost) component in "B1" on level 0.

Therefore, we now collect the label equivalence pairs between adjacent levels (line 10). The same boundary-walking operation is performed as in *Phase I*, except it pairs cells on a given level with neighboring cells on the next-coarser level, rather than those on the same level as before. These cross-level equivalence pairs are merged in the global union-find structure (line 11). Because equivalence pairs detected between adjacent levels are all recorded in the same global union-find structure, components spanning more than two levels will be detected, as the union operation is transitive.

The global union-find structure is now complete, with all labels in each component having been merged together. At the end of *Phase II*, provisional labels throughout all boxes are replaced with final labels via lookups in the global union-find structure (line 12).

B. In Situ AMR-aware Connected Component Labeling

We now turn our focus to designing a viable parallel, *in situ* connected component labeling method. In contrast to the above algorithm, the challenge for an *in situ* approach is in efficiently handling connected components that span not only multiple levels and boxes, but also multiple *parallel processes*. Furthermore, due to various load-balancing techniques employed by simulation codes, our method cannot assume any particular distribution of AMR data. Thus, our key strategy is to design a low overhead communication pattern, which enables parallel, *in situ* synchronization of component connectivity information distributed across many processes. Thus, our key goal is to design a low-overhead communication pattern, which enables parallel, *in situ* synchronization of component connectivity information distributed across many processes under any arbitrary data layout.

To achieve this goal, we propose an AMR-aware communication strategy, as depicted in Figure 3, with more detailed pseudocode in Algorithm 2. Similar to Algorithm 1, our *in situ* approach proceeds in two phases. However, this time, our approach emphasizes the inter-process communication, and employs the *ghost cell* communication primitive (which copies labels from the boundaries of adjacent boxes on the same or next-coarser level), which is available in most AMR applications. The first phase detects level-wise components (components residing entirely within one AMR level) by exchanging ghost cell labels between AMR boxes within each level, followed by a gather/scatter of lightweight connectivity metadata. The second phase then joins these level-wise components into global components using an inter-level ghost cell label copy, together with another connectivity gather/scatter. This resolves inter-level "bridge" scenarios where two level-wise components are (only) connected via component(s) on finer and/or coarser level(s).

Algorithm 2: *In situ* connected component detection

```
   Input  : Total AMR levels, N
   Input  : Cell-classifying predicate, ψ
   Input  : Distributed AMR data on each process, D
   Output: Labelled AMR data on each process, L
1  p = get_current_proc_rank
2  //I. Level-wise detection phase
3  for ℓ = {0, ..., N−1} do
4      all processes that have data at level ℓ (in parallel) do
5          UF_ℓ[p], L_ℓ[p] = local_labeling(D_ℓ[p], ψ)
6          offset_labels_within_level(UF_ℓ[p], L_ℓ[p])
7          exchange_ghost_cell_labels(L_ℓ[p])
8          LP_ℓ[p] = collect_label_equiv(L_ℓ[p])
9          aggregate_to_group_leader(UF_ℓ[p], LP_ℓ[p])
10         if p is group leader then
11             UF'_ℓ = set_label_equiv(LP_ℓ, UF_ℓ)
12             send_UF_to_master(UF'_ℓ)
13         UF'_ℓ[p] = spread_UF_to_every_proc_in_group
14         update_local_labels(L_ℓ[p], UF'_ℓ[p])
15 //II. Global detection phase
16 for l = {1, ..., N−1} do
17     fill_ghost_cell_labels_upward(L_ℓ[p], L_ℓ−1[p])
18     all processes that have data at level ℓ (in parallel) do
19         LP'_ℓ[p] = collect_label_equiv(L_ℓ[p])
20         aggregate_to_group_leader(LP'_ℓ[p])
21         if p is group leader then
22             send_label_equiv_to_master(LP'_ℓ])
23 if p is master proc then
24     GUF, GLP = recv_UF_label_equiv
25     GUF' = set_label_equiv(GLP, GUF)
26     distribute_UF_to_group_leaders(GUF')
27 for ℓ = {0, ..., N−1} do
28     all processes that have data at level ℓ (in parallel) do
29         if p is group leader then
30             UF''_ℓ = recv_UF_from_master_proc
31             broadcast_UF_in_group(UF''_ℓ)
32         UF''_ℓ[p] = spread_UF_to_every_proc_in_group
33         update_local_labels(L_ℓ[p], UF''_ℓ[p])
34 return L
```

1) Level-wise Detection Phase: As in the serial version, we first compute box-local connected components on each level. The difference is that now, this step is performed independently on every process. As a result, this invocation yields, for each process p, a union-find structure $UF_\ell[p]$ and provisional component labels $L_\ell[p]$ defined over each box (line 5 of Algorithm 2).

After identifying box-local components, our algorithm proceeds to coordinate components across the whole level. The challenge we face is that a level-wise component could span multiple processes. Furthermore, because the distribution of boxes across processes is unpredictable, these could be any processes. The naïve approach would be to aggregate all label data to one central process, which could then compute level-

wise components with full context. However, this is clearly infeasible in a large-scale, *in situ* environment where communication is expensive and memory is limited; a distributed method is needed.

Thus, we instead adopt a more refined communication process. We base our strategy on a limited exchange of labels via *ghost cells*; that is, each process sends only the labels along the outer boundary of each local box, and only to processes with adjacent boxes. This greatly reduces data transfer, and also limits communication pairs to far fewer than those in an all-to-all collective (especially if simulation load balancing takes box locality into account [18]). Additionally, AMR simulations typically already have ghost cell communication primitives available; for instance, Chombo exposes the *exchange* function for this purpose.

Once ghost cell labels have been exchanged (line 7 of Algorithm 2), the next step is to relate local labels across processes. To do this, each process p walks the boundaries of its local boxes, building a list of label equivalence pairs $LP_\ell[p]$ between the local boundary cells and surrounding ghost cell labels (line 8). These ghost cell labels represent adjacency information from other processes, and so these equivalences are essentially *union operations* that can join labels on different processes.

However, these label equivalences must be resolved in a level-wide context to be meaningful. Therefore, the next step is to elect a "level leader" process, to which all processes send their local $LP_\ell[p]$ and $UF_\ell[p]$ to be aggregated (line 9). These metadata structures are very small, so communication and memory costs are low. At the level leader, all $UF_\ell[p]$ are merged into a single union-find structure UF_ℓ for the whole level (still line 9), and all label equivalences $LP_\ell[p]$ are applied, finalizing the union-find structure as UF'_ℓ (line 11). The level leader then partitions UF'_ℓ into updated union-find structures $UF'_\ell[p]$ and scatters to each process its pertinent portion (line 13). Lastly, each process applies its new $UF'_\ell[p]$ structure to update all local labels (line 14). At this point, all labels have been resolved in level-wide context, and so level-wise components have been computed. Note: line 12 is intentionally skipped here, and will be revisited in the next phase.

We now clarify one detail that was glossed over previously. Generally, ghost cell exchange only copies cell *values* (labels, in our case), but does not indicate *source processes*. If the labels exchanged are processor-local (i.e., 0-based for each process), it is impossible to differentiate labels from different processes. Therefore, before the ghost cell label exchange, all processes perform an "MPI_Scan" to communicate their local label counts and offset their labeling to a processor-unique ranges (line 6). This way, no ambiguity exists when labels are exchanged.

2) Global Detection Phase: At this point, all connected components have been expanded to a level-wide context. However, as stated before, it is still possible for larger connected components to exist that span multiple AMR levels. Therefore, connectivity information must now be shared between levels. Once again, we leverage ghost cell communication, as AMR simulations will generally also support cross-level ghost cells populations. However, normally such cross-level ghost cells are filled using some form of interpolation, in order to map field values from coarser cells to the finer cells. For instance, Chombo supplies a $fillInterp$ primitive for this purpose, and we use this function to transfer the labels.

Thus, sharing inter-level connectivity begins with performing a ghost-cell copy of box labels from each level to next-finer level (line 17 of Algorithm 2). After populating the ghost cell labels, each process p performs the same box boundary walk as in the level-wise detection phase to produce a new list of label equivalence pairs $LP'_\ell[p]$ (line 19). This time, however, the equivalences relate components on different levels, rather than different boxes on the same level. $LP'_\ell[p]$ are then sent to the same "level leaders" elected before, forming level-wise LP'_ℓ (line 20).

In order to reconcile inter-level label equivalences LP'_ℓ, they must be gathered to a single process to achieve global context. Therefore, a "master" process is elected from the level leaders; LP'_ℓ are then further aggregated to the master (line 22). Also, back on line 12, which we glossed over during the previous phase, the level leaders sent their finalized level-wise union-find structures UF'_ℓ to the master. Thus, at this time, the master will receive and concatenate level-wise union-find structures UF'_ℓ into a global GUF *and* level-wise label equivalences LP'_ℓ into a global GLP (line 24).

The master then applies all label equivalences GLP to GUF (line 25), and distributes to each level leader its respective portion UF''_ℓ of GUF (line 26). These union-find structures are further partitioned and distributed to each process p as $UF''_\ell[p]$ (lines 30 through 32), which are then used to update all local labels once more (line 33). At this point, all labels on all processes have achieved global context and are fully finalized, representing the maximal global connected components in the AMR structure.

Similar to the first phase, relating labels across multiple processes is more nuanced than first described. Multiple levels may have a given label x, and these labels are unrelated. A label-offsetting operation must be performed, as before. However, this time we have a more efficient option besides MPI_Scan. The block-structured AMR model we consider enforces a "proper nesting" constraint, which means cells at level ℓ may never adjoin any cell on level $\ell-2$ or less. Thus, ghost cells in this phase necessarily come from only the next-coarser level. Since all labels are level-relative, and the source level is known, no ambiguity exists. Therefore, when the master receives level-wise UF'_ℓ and LP'_ℓ, it offsets the labels using this knowledge, avoiding an extra round of communication.

3) Hierarchical Group Computing: The methodology described thus far makes the assumption that all processes participate in every step of the algorithm. However, a key observation is that not all processes have data (boxes) for every AMR level. This is especially true for coarser levels, which have relatively few boxes to distribute. Therefore, such

processes would be idle when participating in a given level, limiting potential parallelism.

Therefore, we refine the communication structure of the algorithm using a hierarchical grouping strategy. Processes are now collected into (potentially overlapping) "level groups," one for each AMR level, where each level group contains all processes that actually have data at the corresponding AMR level. Each "level leader" discussed in the previous section is elected from the corresponding level group, collectively forming the "leader group." Finally, the "master" process is elected from the leader group. In our implementation, level leaders and the master are chosen arbitrarily from their respective groups. A more sophisticated strategy could take network placement, etc. into account; we leave exploration of this topic as future work.

The key feature of this strategy is that communication for each level is restricted to the corresponding level group. In Algorithm 2, lines 4, 18, and 28 demonstrate this optimization. This has the effect of limiting the scope of collective communications, which potentially reduces overall communication cost and network contention at the level leaders and the master. This has the additional consequence of allowing processes *without* data at a given level to begin work on the next level without blocking.

Finally, the hierarchical grouping allows a computational optimization, as well. During the global detection phase, inter-level label equivalence pairs generated by processes are first aggregated to the level leaders before being forwarded to the master. Duplicate pairs may be initially generated when a large component on one level adjoins another large component on the next coarser/finer level, as many processes may report the same equivalence. With the hierarchical grouping, the level leaders can remove these duplicates before forwarding to the master, parallelizing the task and reducing communication.

IV. Experimental Evaluation

To evaluate our connected component detection methodology, we collect experimental results using two AMR simulations. The first, BISICLES, is a large-scale AMR ice sheet modeling code for climate modeling. Using this application, we demonstrate the performance of our connected component detection under two real-world simulation configurations. Second, we use a "packed channel" benchmark to test scalability and collect timing breakdowns for our method. In both cases, we apply our method in an *in situ* context, operating on AMR data distributed across many parallel processes.

A. Experimental Setup

All experiments were conducted on the "Edison" supercomputer at the National Energy Research Scientific Computing Center (NERSC). Edison consists of 5,576 compute nodes, each with two 12-core 2.4 GHz Intel "Ivy Bridge" processors and 64 GB memory.

We integrate our connected component detection algorithm directly with the Chombo block-structured AMR framework [13] (specifically, a development version of Chombo 3.2). Chombo is a popular AMR framework used by many scientific applications [4], [14], [15]. Both BISICLES and the packed channel example are Chombo-based; our experiments with these codes demonstrate our Chombo integration.

B. BISICLES – Real-Time Ice Calving Detection Use Case

The BISICLES AMR code models ice sheet dynamics in regions such as Antarctica and Greenland. Connected component detection is of particular interest in BISICLES, as this algorithm could be used to detect *ice calving events* that occur during the simulation. Ice calving occurs when a large iceberg breaks from the main floating ice shelf and floats on its own. Large-scale calving events (e.g., an iceberg of size comparable to the city of Atlanta breaking off the edge of a glacier) are of *scientific interest*, e.g., in studying global climate change [19], [20]. Furthermore, calving events also impact the *correctness* of a BISICLES simulation. If a calving event produces a section of disconnected ice which is not detected and removed in real time, the simulation's underlying mathematical model becomes ill-posed and will fail. Thus, BISICLES relies on real-time connected component detection to identify and remove isolated icebergs.

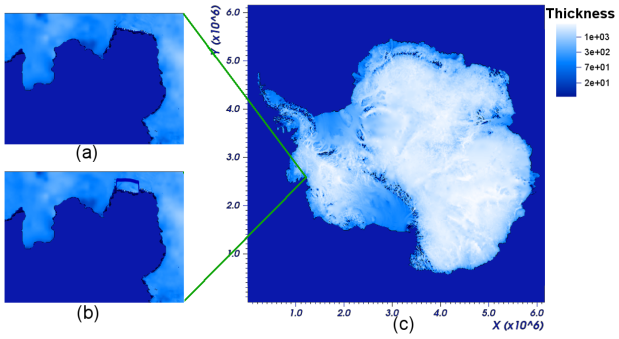


Fig. 5. An example of an ice calving event on the Pine Island Glacier ice shelf during the simulation run is shown. The glacier is shown before (subfigure (a)) and after (subfigure (b)) the calving event occurs, wherein a rectangular region of ice (approximately 30km × 26km in real size) detaches from the main shelf and becomes an isolated floating iceberg. Subfigure (c) shows an ice thickness mapping over the whole of the Antarctica glacier, surrounded by ocean (dark blue).

Figure 5 shows an example of an ice calving event at the Pine Island Glacier ice shelf, with images before (subfigure (a)) and after (subfigure (b)) the calving event. An ice calving event occurs when a region of ice meets two criteria. First, the region must be separated from the main ice shelf by ice that falls below some thickness threshold; this maps exactly to the connected component detection problem with a cell-classifying predicate based on ice thickness. Second, such an *isolated* region must also be fully *floating*, meaning that no part is *grounded* (i.e., sitting on solid ground underneath the ice); this latter condition can be trivially checked for any isolated region. Thus, the challenge is in detecting new connected components in real time, and this is where our method comes in.

We conduct two experiments with BISICLES. First, we

consider a BISICLES run that includes a large ice calving event, which we use to demonstrate detection and to measure performance. The second configuration proceeds for many timesteps with *no* calving events; this shows that our method performs well even when no calving is occurring (relevant because ice calving is rare on a timestep-by-timestep basis).

1) Performance with an ice calving event: We conduct this experiment on the Antarctic continental ice sheet, which plays a vital role in global oceanic and climatic systems. The simulation begins with a base resolution of $8km$ and generates finer AMR levels up to a finest resolutoin of $500m$ using refinement ratios of 2. Our detection algorithm is invoked every timestep during the simulation run, using a cell-classifying predicate of *ice thickness* > 1.0 (in other words, cells with ice thickness under 1 meter are considered to be empty of ice). Whenever our method detects a new connected component, we perform a simple scan of the "groundedness" field over the component to determine if it is floating; if so, it is reported to the simulation for proper recording and handling.

We compare the performance of our method, which we term *PCCL* for Parallel Connected Component Labeling, with the performance of an existing specialized ice calving detection method currently implemented in BISICLES, which we refer to as *Ad Hoc 1*. The *Ad Hoc 1* algorithm relies on an iterative method to find connected components. The data is repeatedly scanned with a sort of flood-fill algorithm, interleaved with several communication rounds using Chombo's ghost cell primitives to enable the fill to cross box boundaries. The loop ends once the detection results stabilize, or once a maximum number of iterations are expended.

Figure 6 shows that our *PCCL* approach consistently outperforms this built-in BISICLES *Ad Hoc 1* method, with greatly increased speedups at larger per-process data sizes. This is because our proposed approach uses less communication, resolving global connectivity using only two communication phases, whereas the iterative *Ad Hoc 1* requires a variable (and potentially much larger) number of communication rounds.

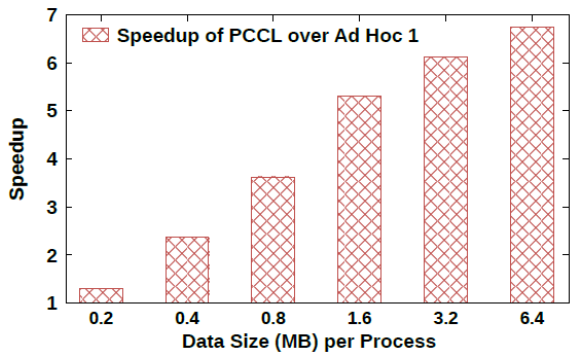


Fig. 6. Speedup of *PCCL* over *Ad Hoc 1* for real-time ice calving event detection in BISICLES.

2) Performance with no ice calving event: Since ice calving is rare on a timestep-by-timestep basis, it is crucial to ensure the detection process does not impose undue overhead during timesteps with no ice calving event. Thus, we also test the performance of both methods on a configuration of BISICLES where no calving occurs for many timesteps.

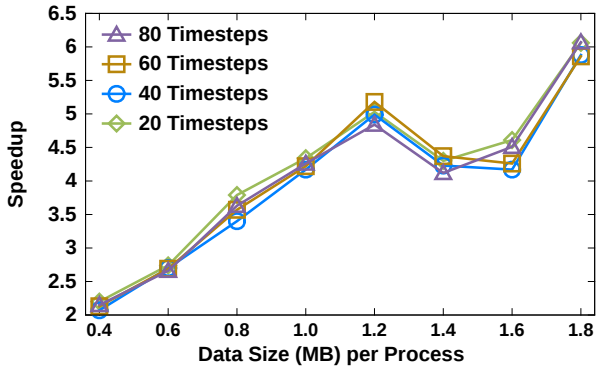


Fig. 7. Speedup of *PCCL* over *Ad Hoc 1* in a BISICLES run where no calving occurs for many consecutive timesteps. Greater speedups are observed at larger per-process data sizes, as before. This speedup is consistent as more no-calving timesteps transpire.

Figure 7 shows the speedup of the *PCCL* method over the *Ad Hoc 1* method under the no-calving configuration. Two trends can be observed from the figure. First, as the data size per process increases, the relative speedup of our method increases commensurately. There is slight dip in speedup around certain data sizes; we theorize that the AMR load balancing algorithm is distributing boxes in a more scattered fashion for these sizes, leading to slightly increased communication. Regardless, PCCL consistently yields speedups over *Ad Hoc 1*, from 2x to 6x. Second, at a fixed data size per process, *PCCL* yields steady speedups (up to 6x) for more consecutive timesteps. The primary reason for these speedups is that our method admits an optimization. Before the global detection phase, candidate components can be quickly checked for viability as calved icebergs by testing the "groundedness" property. If all components on all levels are grounded, *Phase II* can be skipped entirely, as no floating components could be produced. This optimization is only made possible by our method; whereas, the *Ad Hoc 1* approach cannot support this optimization, as no components are available to check until the very end of its execution.

C. Packed Channel – Large-scale Chombo Benchmark

We now turn to a simulation benchmark – the packed channel [16], [17], for further performance evaluation. This simulation unsurprisingly models a "packed channel," which is an idealized approximation of realistic pore spaces for porous media flow obtained from image data of laboratory experiments. Numerical experiments can be run in these channels, which exhibit similar properties to the experimental medium such as porosity, tortuosity, and heterogeneity, effectively mimicking the natural material without the time-consuming process of imaging a real experiment. In turn, the results of this numerical simulation can be used to predict bulk parameters such as permeability, dispersivity, and reaction rates for better continuum scale models. We choose the packed channel for this set of experiments because the obstructions

(filled circular regions) modeled within the channel form well-defined connected components, making them ideal candidates for detection with our algorithm (see Figure 8).

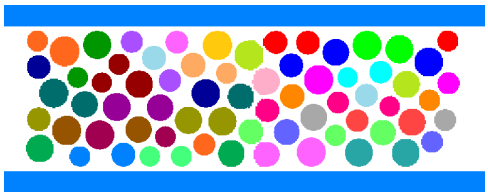


Fig. 8. A depiction of a 2D packed channel dataset. The circular regions in the channel represent obstructions, with the remainder consisting of "pore space" through which fluid may flow. We color the circular regions, showing the connected components our method aims to detect.

Data preparation for the packed channel proceeds as follows. First, the initial data is generated on a uniform mesh. We then apply a second tool to refine the grid at and around the boundaries of the circular obstructions, to increase the resolution around these intricate surfaces. Next, the AMR mesh data is partitioned using Chombo's load-balancing mechanism ("LoadBalance"), spreading the data across parallel processes. Finally, we invoke the connected component detection algorithm, which runs in a parallel, *in situ* context, exactly as if the data were generated in real time.

As stated in Section I, to our knowledge, no general-purpose connected component detection method exists for AMR data. BISICLES provides a specialized ice calving detection routine (*Ad Hoc 1*), however it actually computes an indistinguishable set of cells representing isolated, floating ice, which may belong to any number of connected components. While this simpler formulation works for ice calving, as *grounded* components are specially eliminated before being returned, it cannot solve problems where each connected component must be separately identified. Nonetheless, *Ad Hoc 1* is the closest analogue to our method we can find for comparison, so we adapt it to the packed channel by removing the "grounding" constraint, terming the modified version "*Ad Hoc 2*." Despite our method being "disadvantaged" by solving the harder, general version of the connected component detection problem, it still outperforms the special-purpose *Ad Hoc 2* in the following results.

Figure 9 shows the performance comparison between *PCCL* and *Ad Hoc 2* on the packed channel dataset ($\approx$ 850 MB data) using a varying number of processes. *PCCL* is shown to achieve a consistent speedup over *Ad Hoc 2*, up to 1.4x. A detailed breakdown of each method's performance is also given, showing three timings:

- "Local Computation" includes core-local operations, primarily the local labeling algorithm (a fixed number of passes for *PCCL*, and iterative passes for *Ad Hoc 2*).
- "Chombo Communication" consists of time to invoke Chombo's built-in ghost cell communications, the *fillInterp* and *exchange* functions.
- "Communication" indicates any other communication induced by the method (only *PCCL* invokes it, for aggregating and spreading label equivalence).

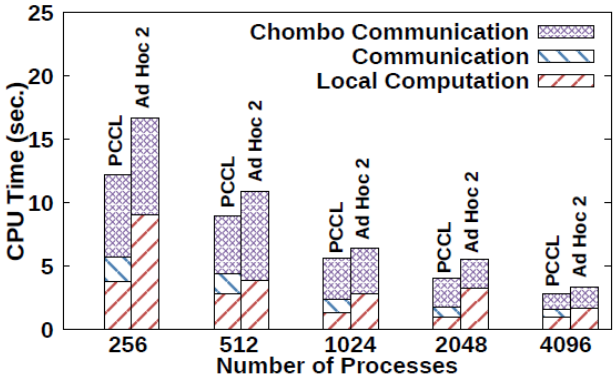


Fig. 9. A timing breakdown comparison between *PCCL* and *Ad Hoc 2* for an AMR dataset with 800 million total cells when run on Edison. *PCCL* achieves a consistent speedup over *Ad Hoc 2*, up to 1.4x.

The "Local Computation" portion of both methods scales well, with *PCCL* exhibiting generally lower local compute times. We attribute this to the relative computational efficiency of a two-pass-based method, versus the multi-pass underpinnings of *Ad Hoc 2*. We see that our "Communication" cost also scales well, which we attribute to our hierarchical grouping strategy that relieves the potential communication bottleneck at the master process.

Finally, we note that the sum of "Communication" and "Chombo Communication" is similar for both *PCCL* and *Ad Hoc 2*, which at first glance is unexpected, as the multi-pass *Ad Hoc 2* ought to invoke much more communication. Upon closer examination, we discover that the iterative *Ad Hoc 2* only requires one iteration to solve this dataset. As a multi-pass-type algorithm, *Ad Hoc 2* is heavily dependent on the specific data, and it turns out that packed channel represents a best-case scenario. In contrast, the realistic BISICLES data used earlier require a larger number of iterations compared to *Ad Hoc 2*, and thus yields higher speedups for our method. We observe that, even under ideal conditions for *Ad Hoc 2*, *PCCL* yields superior overall performance.

V. Related Work

Connected Component Labeling (CCL) on uniform meshes has been explored in both serial and parallel contexts. The main trend in the serial CCL research is to develop a fast way of propagating a component's connectivity throughout the grid. Proposed approaches generally fall into one of three categories, based on the method of connectivity propagation.

Multi-pass algorithms [12] perform "sweep" operations (scanning the dataset and connecting labels via immediate neighbors) in alternating directions until the labeling becomes stable. An inherent drawback is that sweep operations could become quite costly when components exhibit certain complex geometries [12].

Two-pass algorithms [8]–[11] combine an initial scan of the data, which updates an auxiliary union-find structure, with a second update scan to convert provisional labels to final labels using the union-find structure. Variants of this approach differ

in the representations for the union-find structure; we use the array-based union-find structure espoused by [8].

One-pass algorithms [5]–[7] perform labeling with a single scan, during which unlabeled cells are identified, followed by a flood-fill-like assignment of the same label to all connected cells. Although termed single-scan, one-pass algorithms can perform worse than a two-pass algorithm when components have complex geometry [8]. Also, one-pass algorithms rely on contour tracing, which is hard to parallelize due to the difficulty of capturing the global geometry of components.

On the parallel front of CCL research, especially in a shared memory environment, the research challenges center around both load balancing and reducing communication. For example, two parallel CCL methods are proposed in [21]. In the first, the mesh is distributed in equal-size pieces to each process, on each of which local labeling computation is performed. Global labels are then computed through a binary tree based merging process wherein boundary labels are communicated. The second proposed method, designed for isosurface extraction, assigns cells to each process based on value ranges. This strategy eliminates the need for connectivity exchange, as components divided by cell values already reside on the same process. Another multi-phase approach is described in [22], and labels a uniform mesh distributed across processes. This approach has four phases: first, identify local connected components; second, build a global labeling across all processes; third, determine which components span which processes; and fourth, merge global labels to produce a consistent labeling across all processes. A last approach is presented in [23], which detects and tracks features (analogy of the connected components) on AMR data across multiple time-steps. In this approach, it first performs local feature detection in parallel while a simulation is running, and then aggregates detected features to an external processing center, "viz-accumulator", where the global features are finally identified.

Two major differences stand between our work and these existing parallel CCL methods. First, existing approaches work on single-level, uniform meshes, whereas our work deals with multi-level, hierarchical AMR meshes for the first time. Second, we address the connected component detection problem in an *in situ* context, as opposed to the post-processing environments considered in existing work.

VI. Conclusion and Future Work

In this work, we formally define, and then solve, the connected component detection problem for block-structured AMR. We then extend our proposed method to further address this problem in the parallel, *in situ* context. By using a multi-phase AMR-aware communication pattern, we are able to efficiently synchronize connectivity information across the AMR hierarchy. We additionally optimize the inter-process communication pattern in our approach by using a hierarchical grouping strategy to avoid expensive all-to-all communication. We argue that in our methodology, processing connected component detection in the maximum parallelism and exchanging components' connectivity with the least amount of data size are general to the connected component detection on the block-structured N-dimensional AMR data.

Our results show that our method can be used to solve the particular analysis problem of ice calving event detection in the BISICLES simulation, outperforming previous special-purpose algorithms used for this task with up to 6.8x speedup. Speedups are also observed (up to 6x) during timesteps when no ice calving occurs. Additionally, we show scalability up to 4,096 cores on the Edison supercomputer, demonstrating the viability of our method for large-scale, parallel, *in situ* use in real-world applications.

VII. Acknowledgment

The authors would also like to thank Stephen Cornford and Ann Almgren for their invaluable help. We would also like to thank the National Energy Research Scientific Computing Center and Oak Ridge National Laboratory for the use of resources. Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under Contract DE-AC05-00OR22725. Support for this work was provided by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and the U.S. National Science Foundation (Expeditions in Computing and EAGER programs). Work at the Lawrence Berkeley National Laboratory was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of Computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[2] M. Berger and A. Jameson, "Automatic adaptive grid refinement for the Euler equations," *AIAA Journal*, vol. 23, pp. 561–568, 1985.

[3] M. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics*, vol. 82, no. 1, pp. 64–84, 1989.

[4] S. L. Cornford, D. F. Martin, D. T. Graves, D. F. Ranken, A. M. Le Brocq, R. M. Gladstone, A. J. Payne, E. G. Ng, and W. H. Lipscomb, "Adaptive mesh, finite volume modeling of marine ice sheets," *Journal of Computational Physics*, vol. 232, no. 1, pp. 529–549, 2013.

[5] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, 2004.

[6] Q. Hu, G. Qian, and W. L. Nowinski, "Fast connected-component labelling in three-dimensional binary images based on iterative recursion," *Computer Vision and Image Understanding*, vol. 99, no. 3, pp. 414–434, 2005.

[7] J. K. Udupa and V. G. Ajjanagadde, "Boundary and object labelling in three-dimensional images," *Computer Vision, Graphics, and Image Processing*, vol. 51, no. 3, pp. 355–369, 1990.

[8] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Analysis and Applications*, vol. 12, no. 2, pp. 117–135, 2009.

[9] T. Gotoh, Y. Ohta, M. Yoshida, and Y. Shirai, "Component labeling algorithm for video rate processing," in *Proc. SPIE 0804, Advances in Image Processing*, 1987, pp. 217–224.

[10] R. Lumia, "A new three-dimensional connected components algorithm," *Computer Vision, Graphics, and Image Processing*, vol. 23, no. 2, pp. 207–217, 1983.

[11] S. Naoi, "High-speed labeling method using adaptive variable window size for character shape feature," in *Asian Conference on Computer Vision*, vol. 1, 1995, pp. 408–411.

[12] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding*, vol. 89, no. 1, pp. 1–23, 2003.

[13] M. Adams, P. Colella, D. T. Graves, J. Johnson, N. Keen, T. J. Ligocki, D. F. Martin, P. McCorquodale, D. Modiano, P. Schwartz, T. Sternberg, and B. V. Straalen, "Chombo software package for AMR applications-design document," 2000.

[14] K. S. Perumalla, R. M. Fujimoto, P. J. Thakare, S. Pande, H. Karimabadi, Y. Omelchenko, and J. Driscoll, "Performance prediction of large-scale parallel discrete event models of physical systems," in *Proc. Water Simulation Conference*, 2005, p. 9.

[15] D. Trebotich, P. Colella, G. Miller, A. Nonaka, T. Marshall, S. Gulati, and D. Liepmann, "A numerical algorithm for complex biological flow in irregular microdevice geometries," in *Technical Proceedings of Nanotechnology Conference and Trade Show*, vol. 2, 2004, pp. 470–473.

[16] D. Trebotich, M. F. Adams, C. I. Steefel, S. Molins, and C. Shen, "High resolution simulation of pore scale reactive transport processes associated with carbon sequestration," *Computing in Science and Engineering*, vol. Nov/Dec Leadership Computing Issue, 2014.

[17] S. Molins, D. Trebotich, C. I. Steefel, and C. Shen, "An investigation of the effect of pore scale flow on average geochemical reaction rates using direct numerical simulation," *Water Resources Research*, vol. 48, no. 3, 2012.

[18] Z. Lan, V. E. Taylor, and G. Bryan, "Dynamic load balancing for structured adaptive mesh refinement applications," in *Parallel Processing, 2001. International Conference on*. IEEE, 2001, pp. 571–579.

[19] W. Dunham, "Scientists monitor huge iceberg that broke off from Antarctica," http://in.reuters.com/article/2014/04/23/us-science-iceberg-idINKBN0D90ZY20140423, April 2014.

[20] V. Linares, "Giant iceberg breaks off Antarctica glacier," http://www.upi.com/Science_News/2013/07/12/Giant-iceberg-breaks-off-Antarctica-glacier/5841373628126/, July 12, 2013.

[21] A. Choudhary and R. Thakur, "Connected component labeling on coarse grain parallel computers: an experimental study," *Journal of Parallel and Distributed Computing*, vol. 20, no. 1, pp. 78–83, 1994.

[22] C. Harrison, H. Childs, and K. P. Gaither, "Data-parallel mesh connected components labeling and analysis," in *Eurographics Conference on Parallel Graphics and Visualization*, 2011, pp. 131–140.

[23] J. Chen, D. Silver, and L. Jiang, "The feature tree: Visualizing feature tracking in distributed amr datasets," in *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. IEEE Computer Society, 2003, p. 14.